プログラミング言語実験・C 言語第三回課題レポート

1610096 牛山奎悟

2018 年 5 月 3 日

1 common.c の関数の処理内容について

1.1 copy_table(int dst_table[8][15], int src_table[8][15])

1. 何の処理が実装されてるか
   関数内のコメントにあるとおり, この関数では渡された配列 src_table の内容を dst_table にコピーする. つまりカードテーブルの情報をコピーする処理が実装されている.
2. 配列をどのように使って処理しているか
   L14: dst_table[i][j]=src_table[i][j]; とあるように, dst_table はコピー先の配列として用いられていて, src_table はカードテーブルの情報が入った source , コピー元として用いられている.
3. 何故, 機能が実現できてるか
   for 文の 2 重ループを要素全てをさわるように (i<8, j<15) 用いることで二次元配列の要素全てを参照することができ, それぞれの要素をコピー先の配列 dst_table に代入しているのでコピーすることができている.

1.2 clear_table(int cards[8][15])

1. 何の処理が実装されてるか
   関数内のコメントにあるとおり, この関数では渡された配列 cards の要素全て 0 にする, つまりカードテーブルの情報をリセットまたは初期化する処理が実装されている.
2. 配列をどのように使って処理しているか
   cards は要素を全て 0 にするための配列として用いられており, それぞれの要素に 0 を代入して処理が行われている.
3. 何故, 機能が実現できてるか
   (1) と同様に for 文の 2 重ループを要素全てをさわるように (i<8, j<15) 用いることで二次元配列の要素全てを参照することができ, L27: cards[i][j]=0; とあるようにそれぞれの要素に 0 を代入して機能が実現されている.

1.3 copy_cards(int dst_cards[8][15], int src_cards[8][15]

1. 何の処理が実装されてるか
   関数内のコメントにあるとおり, この関数では渡されたカードテーブルの配列 src_cards のカード情報の部分のみをコピーする処理が実装されており,(1),(2) では, カード情報以外の情報も含めたカードテーブルの情報をコピーしていた.
2. 配列をどのように使って処理しているか
   (1) と同様に, 配列 dst_cards はコピー先の配列として用いられていて, src_table はカードテーブルの情報が入ったコピー元として, 用いられている. また, src_cards[0][0] ~cards[4][14] までの情報がその添字に対応する配列 dst_cards にいれられている.
3. 何故, 機能が実現できてるか
   これも,(1) と同様で, 1 つめの for 文の条件式が i<8 から i<5 に変わっているだけ (cards[5][0] ~は場の状態を示すため) で, それのおかげでカードテーブルのカード情報のみを抽出することが可能になっている.

1.4 clear_cards(int cards[8][15])

1. 何の処理が実装されてるか
   関数内のコメントにあるように, 渡されたカードテーブルのカード情報の部分のみを 0 にし, それが場のカードテーブルだったら, 流れたことを意味し, 手札のだったらなにもカードを持っていない状態にできる.
2. 配列をどのように使って処理しているか
   L54: cards[i][j]=0; とあるように配列 cards におけるカードの情報を示す部分 (cards[0][0] ~cards[4][14]) に対して 0 が代入されている.
3. 何故, 機能が実現できてるか
   (2) と同様に, 基本的には,for 文の 2 重ループを回しているからで, 条件式を i<8 ->i<5 に変えることでカード情報の部分のみに 0 が代入されるように制御できている.

1.5 diff_cards(int cards1[8][15], int cards[8][15])

1. 何の処理が実装されてるか
   関数内のコメントにあるように, 引数として渡されたカードテーブルの配列 cards2[8][15] に含まれるカードを配列 cards1 が持っているかどうか関係なく削除の処理, つまり 0 を代入する処理が実装されている.(ただし,joker を除く)
2. 配列をどのように使って処理しているか
   配列 cards2 に対して, カード情報の部分をなぞっていき, もし 1 があればそれを持つ配列の添字に対応する cards1 の要素に 0 を代入している.
3. 何故, 機能が実現できてるか
   cards2 があるカード持っているというのはカードテーブルにおけるカード情報の部分に 1 がたってい

るということで, またその逆で 0 だということはそのカードを持っていないということだから,1 がたっている部分を見つけたらそのカードの情報に対応する cards1 の要素を 0 にすることで cards2 にあるカードを cards1 から削除するということが実現できている.

1.6 or_cards(int cards1[8][15], int cards2[8][15])

1. 何の処理が実装されてるか
   関数内のコメントにあるように, 引数として渡された配列 cards2 が持っているカードをもう一方の配列 cards1 に加えるというもので, つまり配列 cards2 のカードの情報の部分において 1 または 2 がたっている場合それに対応する cards1 の部分に 1 を代入するという処理が実装されている.(ただし,joker が含まれる)
2. 配列をどのように使って処理しているか
   配列 cards2 のカード情報の部分を 2 重ループでなぞっていき, それぞれの要素をみたとき 0 より大きかったら 1 を対応する cards1 の要素に代入するという処理をしている.
3. 何故, 機能が実現できてるか
   (5) ではカードを削除するために 0 が代入されていたが, カードを持たせてあげるためには 1 を立ててあげれば良いため,1 を代入することで機能が実現できている.

1.7 and_cards(int cards1[8][15], int cards[8][15])

1. 何の処理が実装されてるか
   ビット演算子の AND(&) のような処理で, 引数として渡された配列 cards1 が持っているカード (つまりカード情報の部分で 1 がたっている部分) のうち, 配列 cards2 が持っているカードだけを残す. つまり, あるカードを持っているか否かをみたとき,cards2 のカード情報の部分が 1 かつ cards1 のカード情報の部分が 1 だった時,cards1 の各要素は 1 を保つことができる処理が実装されている.
2. 配列をどのように使って処理しているか
   配列 cards1 の cards1[0][0]~cards1[4][14] と配列 cards2 の cards2[0][0]~cards2[4][14] を同時に同じ添字の要素をみていき, もし双方が 1 だった場合,1 を代入し, もしそうでなかったら 0 を代入するという処理をしている. つまり cards1,cards2 として渡されたカードテーブルの情報のカード情報を参照するために配列 cards1,cards2 は使われている.
3. 何故, 機能が実現できてるか
   カードを残したい場合は 1 を代入つまりカードを持った状態を保持し, 条件が満たされなかった場合は 0 を代入することでカードを持っていない状態を保持または, カードを削除するという処理がされているため機能が実現できている.

1.8 not_cards(int cards[8][15])

1. 何の処理が実装されてるか
   この関数では, ビット演算子の NOT(~) のような処理 (有無を反転するという意味合いで) がされてい

て, 引数として渡された配列 cards があるカードをもっていた場合はそれを削除し, 持っていなかった場合はそのカードを加えるという処理をしている.

2. 配列をどのように使って処理しているか
   cards[i][j] と 2 重ループでカード情報の部分を参照していき,1 つまりカードを持っていたら 0 を代入し削除,0 つまりカードを持っていなかったら 1 を代入してカードを加えるという処理をしている.
3. 何故, 機能が実現できてるか
   1 だった場合は 0 を, 0 だった場合は 1 を代入することでカードを持っている状態, 持っていない状態を反転することができるため, 機能が実現できている.

1.9 get_field_state_form_field_cards(int cards[8][15], state *field_status)

1. 何の処理が実装されてるか
   場に出されたカードのテーブルを配列 cards として受取りそれをみて, 何のカードが出されたのか, 階段や枚数のしばりはあるのかの情報を field_status にいれ状況を把握する.
2. 配列をどのように使って処理しているか
   カードテーブルの配列 cards を while 文を用いて数の順 (3,4,5,...) で各要素を見ていき, 出されたカードを発見したときループを抜け, 抜けた時 (i, j) の示すカードを持っているから,card[i][j+1] をみることで階段かどうか判断している. 階段だった場合,cards[i][j] から 1 がたっている要素の数を数えて階段数を調べて, 反対に階段ではない時複数枚同じカードが出されている可能性があるため, L132: for(;i<5;i++) とあるように, cards[i][j] から i を動かし柄に関してカードの有無を見ていくことで, 出された枚数を数えている.
3. 何故, 機能が実現できてるか
   階段だった場合 (cards[i][j+1]>0 だった時) field_status->is_sequence に 1 を代入し階段数を数え field_status->order に階段の中の最小値を代入し強さを知ることができ (革命が起きている時は, 階段内の最大値を代入),field_status->suit[i] に 1 を代入し柄を把握でき, 逆に階段ではなかった場合, 枚数を数えるため, そのときは,field_status->suit[i] に対応する柄のカードを配列 cards が持っていた場合 1 を代入していき出されたカードの柄を全て把握することができ, このとき field_status->order に j を代入しカードの強さを代入している (そもそもカードが見つからなかった場合は最小値 0(革命が起きているときは 14) を代入). 最後に,field_status->quantity に count を代入し枚数を記憶 (count>0 なら場にカードがあるということだから,is_no_card を 0 にする). 以上のようにカードテーブルの配列 cards を用いて field_status の各メンバの値を定めることができるため場に出たカードの情報を得るという機能が実現できている.

1.10 get_field_state_from_own_cards(int cards[8][15], state *field_status)

1. 何の処理が実装されてるか
   手札のカードテーブルから field_status を得る関数で, 手札の枚数, プレイヤーのランク, シートの位置, ジョーカーをもっているかや革命かどうか, しばりかどうかの情報も得られる.
2. 配列をどのように使って処理しているか

主に, 引数として渡された手札のカードテーブルの配列 cards の cards[5][]~cards[7][] の部分カード情報ではない部分が用いられる. その要素に 1 がたっているかどうかや, そのまま値を status のメンバに代入するために使われ処理されている.

3. 何故, 機能が実現できてるか
   カードテーブルには, カード情報だけではなく, 革命が起きているかやプレイヤーのランクなど, 場やプレイヤーの情報が入っているためそれぞれの情報を参照し, それに応じて field_status をセットすることで機能が実現されている.

1.11 remove_low_card(int cards[8][15], int num, int rev)

1. 何の処理が実装されてるか
   出せないカードは引数として渡された配列 cards 内で 0 にセットしてしまうという処理.
2. 配列をどのように使って処理しているか
   基準値 num から下 (または上) の配列 cards の部分は全て 0 を代入するというような処理をしている.
3. 何故, 機能が実現できてるか
   引数として, 基準値 num, 革命が起きているか否かの rev を受け取ることで革命が起きていない (rev = 0) なら num 以下の数つまり cards[][num]~cards[][0] を 0 にして, 革命が起きている (rev = 1) なら,num 以上つまり cards[][num]~cards[][14] を 0 にしているため出せないカードは 0 にセットするという機能が実現できている.

1.12 remove_suit(int cards[8][15], int suit[5], int flag)

1. 何の処理が実装されてるか
   マークのしばりが発生している時, 出すことのできないマークのカードの部分を 0 にする.
2. 配列をどのように使って処理しているか
   suit[i]=0 だったとき引数として渡されたカードテーブルの配列 cards[i][] の各要素に 0 を代入するという処理をしている.
3. 何故, 機能が実現できてるか
   引数として suit[5], flag があり前者には出せるマークがセットされていて, 後者はしばりが起こっているかどうかのフラグがセットされている. そして,(suit[i]+flag)%2 により,suit[i] が 0 のとき cards[i][0]~cards[i][14] に 0 が代入される. こうして出せないマークのカードの部分を 0 にすることができる.

1.13 count_cards(int cards[8][15])

1. 何の処理が実装されてるか
   引数として渡されたカードテーブルの配列 cards に含まれているカードの数を求める
2. 配列をどのように使って処理しているか
   cards[i][j]!=0 として各マークと数のカードが含まれているかどうか評価するのにつかわれている.

3. 何故, 機能が実現できてるか
   カードが含まれている場合, カードテーブルの配列の対応する部分が 1 になっているから.

2 select_cards.c の関数の処理内容について

2.1 select_change_cards(int out_cards[8][15],int my_cards[8][15],int num_of_change)

1. 関数では何が実装されてるか.
   大富豪または貧民がカードを渡すときに適したカードを選ぶ機能が実装されている.
2. 配列をどのように使って処理しているか.
   引数としては手札のカード配列 my_cards, サーバー側に渡すための配列 out_cards, 渡すカードの枚数 num_of_change が渡されている. 一時的に操作するための配列として one_cards[8][15] を定義 (clear_table() で初期化),search_low_card() で my_cards の中のカードのうち joker を除いて小さいほうからカードを一枚選び,one_card にコピーする. そして,diff_cards でコピーしたカードを my_cards から削除する, 最後に or_cards で one_card が持っているカードをコピーする. これを num_of_change 回だけ繰り返す. つまり,one_card は my_cards から抽出したカードを記録しておくために使われ,out_cards はサーバー側に渡すために最終的なカードの状態を覚えるために使われている.
3. 該当するソースコードの記述で何故機能が実現できているか.
   search_low_card で手札のうち最も小さいカードを選ぶことができ,diff_cards,or_cards で提出配列に渡すカードをまるで移したかのような処理が可能になっているから.

2.2 select_submit_cards(int out_cards[8][15],int my_cards[8][15], state *field_status)

1. 関数では何が実装されてるか.
   手札から場に出すカードを選ぶ機能が実装されている.
2. 配列をどのように使って処理しているか
   (1) と引数の配列は同じ意味.field_status で場にカードがあるか否かを把握できる. また,out_cards にコピーするために一時的に情報を覚えておく配列 selectcards を定義する. そして,elect_cards_free(), select_cards_restrict(), select_cards_free_rev(), select_cards_restrict_rev() を用いて select_cards に出すカードを手札からコピーし, 最終的に,copy_table() を用いて,select_cards の状態を out_cards にコピーしている.
3. 該当するソースコードの記述で何故機能が実現できているか.
   field_status を使うことにより, 現在場で革命が起きているかどうか, 場にカードがでているかどうか, しばりなどがあるかどうか判別することができ, それに応じてカードを選ぶ関数を変えることができているため, この機能が実現できている.

2.3 select_cards_free(int select_cards[8][15], int my_cards[8][15], state *field_status)

1. 関数では何が実装されてるか.
   何もしばりなど無いとき, 手札の最も弱いカード (joker を除く) を一枚 select_cards にコピーする機能

が実装されている

2. 配列をどのように使って処理しているか
   手札とコピーする先の配列を渡され, 手札の配列 my_cards の中から最も弱いカードを探しそれを select_cards にコピーするため serch_low_card() に渡している.
3. 該当するソースコードの記述で何故機能が実現できているか.
   search_low_card() で手札のうち最も弱いカードを選択することができるから.

2.4 select_cards_restrict(int select_cards[8][15], int my_cards[8][15], state *field_status)

1. 関数では何が実装されてるか.
   (3) とは逆にしばりがあるときに出せるカードを手札から選ぶ機能が実装されている.
2. 配列をどのように使って処理しているか
   同じように渡された配列と自分の手札の代わりに操作をする手札のコピーの配列 tmp_cards を用いて, 階段なら階段の, 柄なら柄の出せないカードを削除する関数に tmp_cards を渡し, 最終的にその中から小さいカードを select_cards にコピーしている.
3. 該当するソースコードの記述で何故機能が実現できているか.
   field_status のメンバの,is_sequence や is_lock, quantitiy, suit, order によって場が階段なのか, ペアのしばりか, 柄のしばりなのか判断することができるからである.